

**Memory arrangement for portable data carrier e.g. chip card, updates information following deactivation process in such way, that referenced memory areas cover joined memory area**

**Patent number:** DE10040241

**Publication date:** 2001-03-22

**Inventor:** CIESINGER DANIEL (DE)

**Applicant:** GIESECKE & DEVRIENT GMBH (DE)

**Classification:**


- international: **G06F12/02; G06K19/073; G06F12/02; G06K19/073;**  
(IPC1-7): G06F12/02; G06F12/16

- european: G06F12/02D2; G06F12/02D2G; G06K19/073

**Application number:** DE20001040241 20000817

**Priority number(s):** DE20001040241 20000817; DE19991039209 19990818

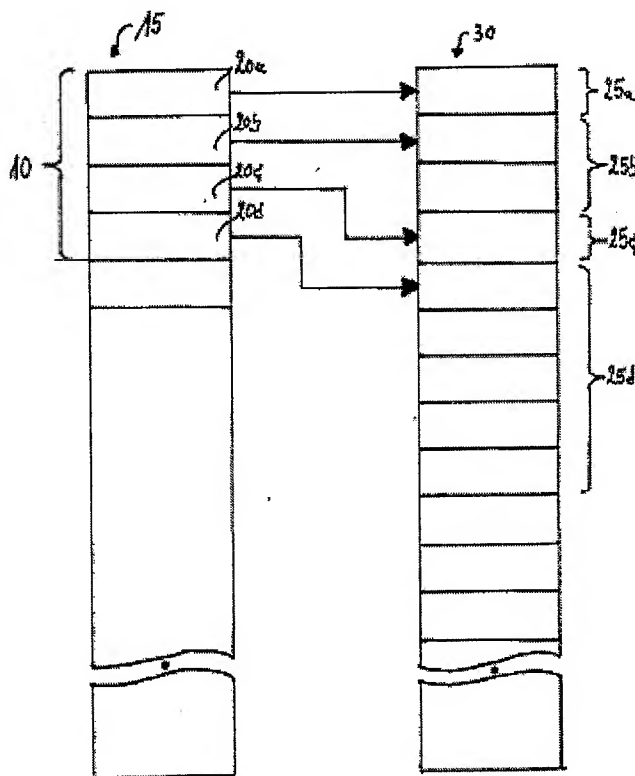
**Also published as:**

 FR2799285 (A1)

[Report a data error here](#)

**Abstract of DE10040241**

The memory arrangement includes a first memory arrangement (30) comprising at least one memory area (10) which stores information (20a, 20b, 20c, 20d) which describes position and size of memory areas (25a, 25b, 25c, 25d) in the second memory arrangement (15) used for the execution of a card program. A program routine updates the information in the memory area following a deactivation process which causes a cancellation of the contents of the first memory arrangement, in such way, that corresponding memory areas in the first memory arrangement cover a joined memory area. An Independent claim is provided for a corresponding memory management method.



Data supplied from the **esp@cenet** database - Worldwide



①9 BUNDESREPUBLIK  
DEUTSCHLAND



DEUTSCHES  
PATENT- UND  
MARKENAMT

⑫ **Offenlegungsschrift**  
⑩ **DE 100 40 241 A 1**

⑤1 Int. Cl. 7:  
**G 06 F 12/02**  
G 06 F 12/16

②1 Aktenzeichen: 100 40 241.0  
②2 Anmeldetag: 17. 8. 2000  
④3 Offenlegungstag: 22. 3. 2001

DE 100 40 241 A 1

⑥6 Innere Priorität:  
199 39 209. 9 18. 08. 1999

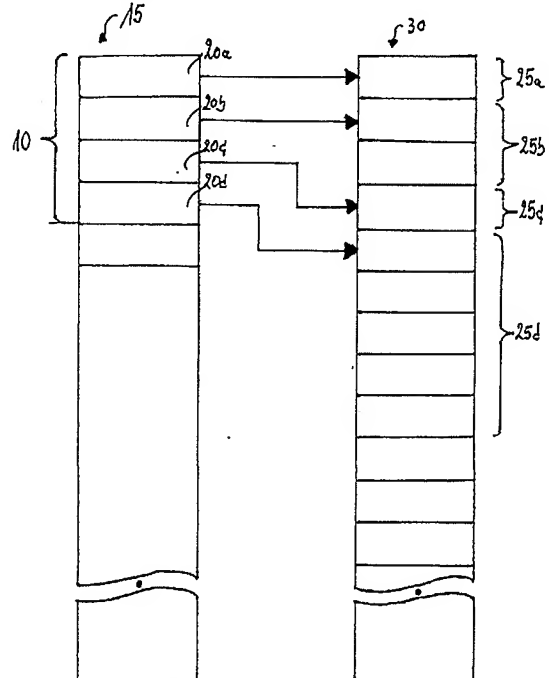
⑦1 Anmelder:  
Giesecke & Devrient GmbH, 81677 München, DE

⑦2 Erfinder:  
Ciesinger, Daniel, 80796 München, DE

**Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen**

⑤4 Speicheranordnung für einen Datenträger und Verfahren zur Speicherverwaltung

⑤7 Vorgeschlagen werden eine Speicheranordnung für einen tragbaren Datenträger sowie ein Verfahren zum Betrieb einer Speicheranordnung. Gemäß einer ersten Ausführungsform umfaßt die Speicheranordnung eine flüchtige erste (30) sowie eine nichtflüchtige zweite Speichereinrichtung (15), wobei die erste Speichereinrichtung (30) mindestens einen Speicherbereich (25) besitzt, dessen Lage und Größe durch in der zweiten Speichereinrichtung (15) abgespeicherte Bereichsangaben (20) bestimmt ist. Jeweils bei einem Ein- oder beim Abschaltvorgang werden die Bereichsangaben in der zweiten Speichereinrichtung (15) neu festgelegt. Gemäß einer zweiten Ausführungsform werden in einem ersten Speicherbereich (80) befindliche aktive Datenobjekte (45b, 45d) selektiv und zusammenhängend in einen freien zweiten Speicherbereich (90) umkopiert und alle verbleibenden inaktiven Datenobjekte (45a, 45c) inaktiviert. Gemäß einer dritten Ausführungsform befinden sich Daten zur Beschreibung des Zustandes einer Methode in einer nichtflüchtigen ersten Speichereinrichtung (100), die die Ausführung der Methode beschreibenden Operativdaten in einer zweiten Speichereinrichtung (200). Bei Aufruf einer zweiten Methode aus einer ersten heraus werden die den Zustand der Methode beschreibenden Daten vorübergehend in eine flüchtige, dritte Speichereinrichtung (300) kopiert.



DE 100 40 241 A 1

## Beschreibung

Die Erfindung betrifft eine Speicheranordnung sowie ein Verfahren zur Speicherverwaltung insbesondere für tragbare Datenträger in Gestalt von Chipkarten.

Während frühe Formen von Identifikationskarten keine maschinenlesbaren Datenspeicher oder allenfalls einen Magnetstreifen zum Speichern und Auslesen geringer Datenmengen aufwiesen, ist es heute möglich, in derartige Karten eingesetzte integrierte Halbleiterschaltungen zu verwenden. Im einfachsten Fall kann diese Halbleiterschaltung als eine mit einer Ein-/Ausgabeeinrichtung versehene Speicherschaltung, beispielsweise einem EEPROM, ausgeführt sein. Für komplexere Anwendungen hat es sich aber als vorteilhaft erwiesen, einen vollständigen Mikrocontroller mit einem Zentraleinheit, einen Speicher sowie Ein-/Ausgabeeinheiten verbindenden Bussystem in einer Karte zu integrieren.

Mit einem Mikrocontroller versehene Chipkarten, sogenannte "Smart Cards", werden in einer Vielzahl von Formen bei einer zunehmenden Vielfalt von Anwendungsbereichen eingesetzt. Üblich sind beispielsweise Karten gemäß der Norm ISO 7810, die aus einem aus Kunststoff gefertigten Kartenträger bestehen, in den eine integrierte Halbleiterschaltung sowie ein Kontaktfeld zum Herstellen elektrischer Verbindungen mit einem entsprechenden Lesegerät eingelassen sind. Daneben sind auch andere Formate im Gebrauch, anstelle der galvanischen Kopplung an ein Lesegerät können insbesondere kontaktlose Signalübermittlungsverfahren treten. Eine Übersicht über bekannte Chipkartentechnologien findet sich zum Beispiel in W. Rankl, W. Effing: "Handbuch der Chipkarten", München: Carl Hanser Verlag, 2. Auflage 1996.

Vorgeschlagen wurde weiter bereits, den Kartenkörper zu verkleinern oder sogar ganz fortzulassen, indem etwa ein Einchip-Mikrocontroller in Armbanduhren, Schmuckstücke, Kleidungsstücke oder andere Gebrauchsgegenstände eingebaut wird. Der Begriff "Chipkarte" soll daher alle derzeitigen und zukünftigen transportablen Gegenstände umfassen, in die ein Mikrocontroller eingebettet ist, um es einem Besitzer oder Inhaber zu ermöglichen, chipkartentypische Interaktionen mit entsprechenden dafür vorgesehenen Interaktionsstationen vorzunehmen.

Wegen der im Verhältnis zu dem für umfangreiche Kartenprogramme benötigten Speicherbedarf begrenzten Speichergröße kommt der Gestaltung und der Betrieb der Speicheranordnungen von Chipkarten eine besondere Bedeutung zu. Übliche Chipkarten besitzen typischerweise einen als EEPROM ausgeführten nichtflüchtigen Speicher für nichttransiente bzw. persistente Datenobjekte sowie einen als RAM ausgeführten, flüchtigen Speicher für transiente Datenobjekte. Aufgrund topographischer Zwänge, Mikrocontroller auf Chipkarten dürfen in der Regel eine Fläche von etwa 25 mm nicht überschreiten, ist der auf einer Chipkarte überhaupt einrichtbare Speicherraum stark begrenzt. Derselbe Grund beschränkt zudem Größe und damit die Leistungsfähigkeit der Zentraleinheit.

Um den vorhandenen Speicherraum bestmöglich zu nutzen, ist eine Fragmentierung möglichst zu vermeiden. Zur Vermeidung der Fragmentierung von Speichern in Datenverarbeitungsanordnungen ist die Verwendung sogenannter "Garbage Collection"-Verfahren bekannt, wie sie etwa in der US 4,907,151 beschrieben sind. Die bekannten Garbage-Collection-Verfahren beruhen dabei überwiegend auf dem Prinzip, den Arbeitsspeicher in vorzugsweise zwei Bereiche zu teilen und für eine Programmausführung zunächst nur einen Bereich zu verwenden, bis in diesem keine ausreichend großen zusammenhängenden Speicherbereiche mehr

vorhanden sind. Ist der benutzte Speicherbereich in diesem Sinne voll, werden durch eine Garbage-Collection-Programmroutine alle aktiven Datenelemente im benutzten Speicherbereich identifiziert und in den zweiten, bis dahin nicht benutzten Speicherbereich umkopiert, wo sie zusammenhängend angeordnet werden.

Voraussetzung für die Ausführbarkeit dieser gängigen Verfahren ist, daß der zur Verfügung stehende Arbeitsspeicher so groß ist, daß die Funktionalität der Prozessoranordnung, deren Teil er ist, durch die Nichtnutzbarkeit jeweils eines Teiles des Arbeitsspeichers nicht beeinträchtigt wird. Chipkarten erfüllen diese Voraussetzung regelmäßig nicht. Ihr aus Raumnot sehr kleiner Arbeitsspeicher muß praktisch immer vollständig den auf der Chipkarte vorhandenen Anwendungen zu Verfügung stehen. Die gängigen Garbage-Collection-Verfahren eignen sich daher für Chipkarten nicht.

Hinsichtlich der Organisation der ein Kartenprogramm definierenden Daten auf einer Chipkarte geht eine gegenwärtige Entwicklung dahin, beim Aufruf einer Methode, einer Prozedur oder dergleichen in der Speicheranordnung des den Programmablauf ausführenden Mikrocontrollers nicht mehr unstrukturierte Datenagglomerationen, sondern strukturierte "Frames" anzulegen. Das allgemeine Konzept von Stapelspeicher-Frames ("Stack Frames") ist am Beispiel der "Java Virtual Machine" unter anderem aus Jon Meyer, Troy Downing: "Java Virtual Machine", Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1997, Seiten 58 und 59, sowie Tim Lindholm, Frank Yellin: "The Java Virtual Machine Specification", Reading, MA, USA: Addison-Wesley, 1997, Seiten 66 bis 68, bekannt. Durch derartige Techniken werden zwar in vielerlei Hinsicht vorteilhafte Chipkarten ermöglicht. Allerdings wird auch die Verwaltung der in der Speicheranordnung angelegten Datenstrukturen komplizierter und ressourcenaufwendiger, insbesondere setzt die bestimmungsgemäße Nutzung solcher Karten einen jederzeit nutzbaren Arbeitsspeicher voraus.

Aufgabe der Erfindung ist es daher, für ein- und abschaltbare tragbare Datenträger, insbesondere in Gestalt von Chipkarten, Maßnahmen anzugeben, die die Nutzbarkeit des Arbeitsspeichers verbessern. Hierzu soll eine im Hinblick auf eine Defragmentierung verbesserte Speicheranordnung sowie ein Verfahren zur Defragmentierung mindestens eines Speicherbereiches einer solchen Speicheranordnung angegeben werden. Desweiteren soll die Möglichkeit geschaffen werden, eine Kompaktierung des Arbeitsspeichers vorzunehmen um möglichst große zusammenhängende Speicherbereiche zurückzugewinnen. Weiterhin sollen eine Speicheranordnung sowie ein Verfahren zum Betrieb einer Speicheranordnung angegeben werden, die bei der Ausführung eines Kartenprogrammes den Aufruf einer zweiten Methode aus einer ablaufenden ersten Methode heraus vereinfachen.

Diese Aufgabe wird gelöst durch eine Speicheranordnung gemäß Anspruch 1 sowie ein Verfahren zur Defragmentierung gemäß Anspruch 5. Die Aufgabe wird weiterhin gelöst durch die in Anspruch 7 angegebene Speicheranordnung sowie das in Anspruch 10 angegebene Verfahren. Die Aufgabe wird weiterhin gelöst durch die in Anspruch 13 angegebene Speicheranordnung sowie das in Anspruch 18 angegebene Verfahren zum Betrieb einer Speicheranordnung.

Vorteilhafte Weiterbildungen und zweckmäßige Ausgestaltungen der erfindungsgemäßen Anordnungen bzw. Verfahren sind den abhängigen Ansprüchen zu entnehmen.

Die Erfindung wird im folgenden anhand von Ausführungsbeispielen exemplarisch näher erläutert. Es zeigen:

Fig. 1 eine vereinfachte Architektur einer Chipkarte,

Fig. 2A die Speicherbelegung einer aus einem EEPROM-Speicher und einem RAM-Speichers gebildeten Speicheran-

ordnung in einem ersten Betriebszustand,

**Fig. 2B** die Speicherbelegung in einem zweiten Betriebszustand,

**Fig. 3A** die Speicherbelegung eines Speichers einer zweiten Ausführungsform einer Speicheranordnung,

**Fig. 3B** die Speicherbelegung eines Speichers einer weiteren Ausführungsform einer Chipkarte,

**Fig. 4A** eine weitere Ausführungsform einer Speicheranordnung mit einem Prozessorregister und zwei Stapelspeichern in einem ersten Betriebszustand,

**Fig. 4B** die Speicherbelegung des zweiten Stapelspeichers bei Aufruf einer zweiten Methode durch eine erste,

**Fig. 4C** die Speicherbelegung der Speicheranordnung nach **Fig. 3A** bei Aufruf einer zweiten Methode durch eine erste.

**Fig. 1** zeigt schematisch und in vereinfachter Form die Architektur eines tragbaren Datenträgers **1**, für den im folgenden die Ausführungsform einer Chipkarte angenommen wird.

Zentrales Element der Chipkarte ist ein Chipkartencontroller mit einer zentralen Prozessoreinheit **2**, welche mit einem ROM-Speicher **3**, einem RAM-Speicher **15**, einem EEPROM-Speicher **30** sowie einer Eingangs/Ausgangsschnittstelle **4** verbunden ist. Der ROM-Speicher **3** enthält das Betriebssystem für die zentrale Prozessoreinheit und wird während der Herstellung der Chipkarte **1** eingebrannt. Der Inhalt des ROM-Speichers **3** ist während der Lebensdauer der Chipkarte unveränderbar. Das Betriebssystem kann insbesondere Programmroutinen zur Verwaltung der aus dem RAM-Speicher **15** und dem EEPROM-Speicher **30** bestehenden Speicherstruktur enthalten. Der RAM-Speicher **15** ist der Arbeitsspeicher der zentralen Prozessoreinheit **3**. Er ist flüchtig, d. h. alle darin abgelegten Daten gehen verloren, wenn die Versorgungsspannung der Chipkarte abgeschaltet wird. Der EEPROM-Speicher **30** ist ein nichtflüchtiger Speicherbereich, in dem Daten oder auch Programmcode unter Kontrolle des Betriebssystems geschrieben und gelesen werden können.

Die Ein/Ausgangsschnittstelle **4** dient zur Kommunikation der Chipkarte **1** mit einer externen Einrichtung sowie zur Spannungsversorgung der Chipkarte. Sie kann kontakthaft, d. h. in Gestalt von Kontaktflächen, oder kontaktlos, d. h. in Gestalt z. B. einer Antenne, ausgeführt sein.

Die Chipkarte ist zur Ausführung von in einer objektorientierten Programmiersprache abgefaßten Kartenprogrammen ausgebildet und gestattet das nachträgliche Laden solcher Kartenprogramme auf die Chipkarte. Kartenprogramme der genannten Art benutzen im Kontext der Programmausführung Datenobjekte, welche sie zu diesem Zweck erzeugen. Die Kartenprogramme selbst liegen dabei im nichtflüchtigen Speicher **15** der Chipkarte, die von ihnen erzeugten Datenobjekte im flüchtigen Speicher **30**.

Die Lage der Datenobjekte im flüchtigen Speicher sowie ihre Größe werden in einem definierten Bereich **10** des nichtflüchtigen EEPROM-Speichers **15** festgehalten. In dem Speicherbereich **10** befinden sich hierfür verschiedene Zeigervariablen **20a, 20b, 20c, 20d**, deren Inhalt jeweils auf den Anfang zugeordneter Speicherbereiche **25a, 25b, 25c, 25d** im flüchtigen RAM-Speicher **30** weist. Die Speicherbereiche **25a, 25b, 25c, 25d** des RAM-Speichers **30** dienen zur Aufnahme der Datenrepräsentationen transients Datenobjekte für ein auf dem Chipkartencontroller ablaufendes Kartenprogramm. Einmal eingerichtete Zeigervariablen und die dadurch bewirkte Aufteilung des RAM-Speichers **30** werden nach Initialisierung eines Kartenprogrammes grundsätzlich beibehalten. Bei Wegfall der Versorgungsspannung gehen daher nur die Inhalte der Datenobjekte im RAM-Speicher **30** verloren, während die Objekte selbst durch die

Zeigervariablen im nichtflüchtigen Speicher **15** erhalten bleiben.

Durch Änderungen im Ablauf eines Kartenprogrammes, Änderung der Struktur eines Kartenprogrammes oder durch Änderung der Zahl der auf der Chipkarte vorhandenen Kartenprogramme kann sich die für Ausführbarkeit der Programme erforderliche Bereichsaufteilung im RAM-Speicher **30** jedoch ändern. Verändert sich beispielsweise die Größe des Speicherbereiches **25a**, kann das Auswirkungen auf die darauffolgenden Speicherbereiche **25b, 25c, 25d** haben. Insbesondere können Speicherbereiche entstehen, auf die kein Kartenprogramm mehr zugreift. Benötigt ein Kartenprogramm etwa nach einer Versionserneuerung zum Beispiel mehr Speicherbereich im RAM-Speicher **30**, kann dieser nur in dem noch nicht benutzten Teil des RAM-Speichers **30** neu zugewiesen werden. Der bisher zugeordnete Speicherbereich wird überflüssig und bildet fortan einen Lückenspeicherbereich. Bei einer Verkleinerung des benötigten Speicherbereiches, etwa des Bereiches **25a**, bestehen die Alternativen, die Speicherbereichsgröße unverändert zu lassen, dadurch aber Speicherplatz zu verschwenden, oder die Verkleinerung entsprechend vorzunehmen, dadurch aber wiederum das Entstehen einer Lücke in der Speicherbelegung zwischen den Speicherbereichen **25a** und **25b** zuzulassen. Ein auf solche Weise entstandener Lückenspeicherbereich kann häufig nicht mehr sinnvoll genutzt werden kann.

Lückenspeicherbereiche werden üblicherweise beseitigt, indem während des Normalbetriebes der betroffenen Datenverarbeitungseinrichtung regelmäßig eine Defragmentierung durchgeführt wird. Die bekannten Techniken zur Defragmentierung von Speichern, in denen sich Datenobjekte von objektorientiert programmierten Kartenprogrammen befinden, setzen allerdings stets das Vorhandensein von ausreichendem Verfügungsspeicherraum voraus. Entsprechender Speicherraum läßt sich aber aus Platzgründen auf solchen Chipkarten nicht bereitstellen. Die bekannten Defragmentierungstechniken lassen sich dadurch auf Chipkarten nicht einsetzen.

Erfindungsgemäß wird deshalb eine Defragmentierung des RAM-Speichers **30** unmittelbar nach dem Ein- oder vor dem Abschalten des Chipkartencontrollers durchgeführt, mithin nach der Beendigung der Ausführung von Kartenprogrammen im Rahmen des Normalbetriebes bzw. vor dessen Beginn. Bei einer zu diesem Zeitpunkt durchgeführten Defragmentierung kann eine Berücksichtigung der Inhalte der in den Speicherbereichen **25a, 25b, 25c, 25d** gespeicherten, transiente Objekte darstellenden Daten entfallen, da der Inhalt des flüchtigen RAM-Speichers **30** ohnehin gelöscht wird. Die Werte der Zeigervariablen **20a, 20b, 20c, 20d** im Bereich **10** im nichtflüchtigen Speicher **15** können daher ohne jedwede Rücksicht auf den flüchtigen Speicher **30** neu eingestellt werden. Der Einschaltvorgang kann etwa durch Verbinden mit einer Betriebsstromquelle oder durch Wiederherstellen der Stromversorgung bewirkt sein, der Abschaltvorgang beispielsweise durch Unterbrechen der Stromversorgung oder durch Trennen von einer Betriebsstromquelle. Zweckmäßig erfolgt eine Neufestlegung der Zeigervariablen bei jeder Inbetriebnahme der Chipkarte. Eine die Neufestlegung durchführende Programmroutine läßt sich, da sie nur vor oder nach Ausführung eines normalen Kartenprogrammes arbeitet, als vergleichsweise kleines Programm darstellen und ist zweckmäßig Teil des Betriebssystems der Chipkarte.

Die Defragmentierung erfolgt, indem die im Speicherbereich **10** abgelegten Zeigervariablen **25a, 25b, 25c, 25d** so bestimmt werden, daß ihre Inhalte im RAM-Speicher unmittelbar aneinandergrenzende Speicherbereiche anzeigen, der RAM-Speicher **30** damit so genutzt wird, daß ein größt-

möglicher zusammenhängender Freibereich im RAM-Speicher 30 entsteht. Fig. 2B veranschaulicht eine danach entstehende Belegung des nichtflüchtigen Speichers 15 einer Chipkarte. Die Inhalte der Zeigervariablen 20a, 20b, 20c, 20d weisen jeweils auf den Anfang zugeordneter Speicherbereiche 25a, 25b, 25c bzw. 25d im flüchtigen RAM-Speicher 30, wobei die Größe der einzelnen Speicherbereiche 25a, 25b, 25c bzw. 25d jeweils genau der von dem Kartenprogramm benötigten Größe entspricht und die Speicherbereiche 25a, 25b, 25c bzw. 25d zusammen einen zusammenhängenden Speicherbereich ohne Lücken repräsentieren.

Fig. 3A zeigt den Speicher 40 einer zur Ausführung von nachladbaren, in einer objektorientierten Programmiersprache abgefaßten Kartenprogrammen ausgebildeten Chipkarte. Kartenprogramme der genannten Art benötigen bei ihrer Ausführung für die Speicherung von im Kontext des Programmablaufes benutzten Datenobjekten in der Regel einen im Vergleich zum RAM der Chipkarte großen, frei belegbaren Verfügungsspeicher. Bei dem Speicher 40 handelt es sich deshalb vorzugsweise um den nichtflüchtigen Speicher der Chipkarte, d. h. üblicherweise um den EEPROM. Die Größe des verfügbaren Speicherraumes wird abgesehen von den Speichermitteln selbst auch durch die Leistungsfähigkeit des Chipkartencontrollers bestimmt. Diese wird durch die beengten Platzverhältnisse ebenfalls begrenzt. In der Regel können Chipkartencontroller deshalb überhaupt nur einen begrenzten Speicherraum adressieren.

Für die Erfindung wird nun ausgenutzt, daß der von einem Chipkartencontroller adressierbare Speicherraum häufig kleiner ist als der physikalisch bereitstellbare. Erfindungsgemäß wird der Speicher 40 deshalb in Form von zwei Speicherbänken 80, 90 ausgeführt, von denen jede zweckmäßig die vom Chipkartencontroller maximal adressierbare Zahl von Speicherplätzen umfaßt. Die Ausführung eines Kartenprogrammes erfolgt jeweils unter Nutzung genau einer Speicherbank. Es sei angenommen, daß zunächst nur die Speicherbank 80 zur Speicherung der Daten verschiedener Datenobjekte 45a, 45b, 45c, 45d im Kontext eines auf dem Chipkartencontroller ablaufenden Kartenprogrammes in Benutzung ist. Aufgrund von Änderungen in der Struktur, der Zusammensetzung oder der Zahl der von der Chipkarte ausführbaren Kartenprogramme können im Laufe der Nutzung zwischen den Datenbereichen der Datenobjekte 45a, 45b, 45c, 45d Speicherbereiche 50a, 50b, 50c mit nicht mehr benutzten Datenobjekten entstehen. Insbesondere kommt es vor, daß einzelne, auch "tot" bezeichnete Datenobjekte 45a, 45c im Kontext des Kartenprogrammes zu keinem zukünftigen Zeitpunkt mehr Verwendung finden werden. Erkennen lassen sich solche toten Datenobjekte 45a, 45c beispielsweise daran, daß das Kartenprogramm über keine Zeiger auf diese Datenobjekte 45a, 45c verfügt.

Um unbenutzten und/oder zerstückelten Speicherplatz in der gerade benutzten Speicherbank 80 erneut für eine Verwendung im Kontext des Kartenprogrammes nutzbar zu machen, wird auf den Eintritt vordefinierter Ereignisse hin oder zu bestimmten Zeitpunkten eine Übertragung der in der Speicherbank 80 abgelegten Daten in die jeweils andere, zuletzt nicht benutzte Speicherbank 90 vorgenommen. Hierzu wird eine Kompaktierungsroutine in Gang gesetzt, welche grundsätzlich einem üblichen Garbage-Collection-Verfahren entspricht. Während der Dauer des Speicherbankwechsels, d. h. während der Ausführung der Kompaktierungsroutine werden keine anderen Kartenprogramme ausgeführt. Die Grundreferenzen bzw. die Grundzeiger auf die angelegten Datenobjekte 45a, 45b, 45c, 45d befinden sich daher außerhalb des fragmentierbaren Arbeitsspeichers 80.

Die Übertragung erfolgt, indem durch selektive Kopiervorgänge 55a, 55b ausschließlich die noch aktiven, auch "le-

bend" genannten Datenobjekte in die zuletzt nicht benutzte Speicherbank 90 umkopiert werden, so daß darin ein zusammenhängender benutzer Teilbereich 60a mit allen lebenden Datenobjekten sowie ein zusammenhängender freier Teilbereich 60b entstehen. Alle toten Datenobjekte verbleiben in der Speicherbank 80. Sind alle lebenden Datenobjekte umkopiert, tauschen die Speicherbänke ihre Rollen, d. h. nachfolgend wird die Speicherbank 90 für die Ausführung von Kartenprogrammen genutzt, die Speicherbank 80 mit den darin verbliebenen, toten Datenobjekten zur vollständigen Neubelegung freigegeben. Eine geeignete Technik zur Auswahl der zu kopierenden lebenden Datenobjekte ist unter der Bezeichnung "Copy-Live"-Verfahren bekannt und beispielsweise in Jürgen Heymann: "Mathematical Modelling and Hardware Support of Garbage Collection", TUM-INFO-06-90-122-350/1 TUM-19022, München: Mathematisches Institut und Institut für Informatik der Technischen Universität München, 1990, S. 30 bis 35, beschrieben. Durch ein solchermaßen durchgeführtes Umkopieren beim Wechseln der benutzten Speicherbank wird ein kombinierter "Garbage Collection"-Defragmentierungsvorgang realisiert.

Fig. 3B zeigt eine Variante des kombinierten Freigabe- und Defragmentierungsvorganges aus Fig. 3A, deren Verwendung vor allem dann zweckmäßig ist, wenn der Speicher 40 mit den Speicherbänken 80 und 90 in einer Technik realisiert ist, bei der Schreibzugriffe besonders viel Zeit beanspruchen, d. h. wenn es sich bei dem Speicher 40 zum Beispiel um einen EEPROM handelt. Um die Zeit für die Durchführung der Defragmentierung und damit die Zeit für den Speicherbankwechsel trotzdem möglichst klein zu halten, ist für den Umkopiervorgang ein RAM-Cache 95 vorgesehen. Darin wird mittels selektiver Kopierschritte 65a, 65b zunächst die defragmentierte Aneinanderreihung der lebenden Datenobjekte 45b, 45c erzeugt. Für jedes kopierte lebende Datenobjekt erfolgt dabei in dem RAM-Cache 95 eine Auflösung und Anpassung der Referenzen auf die anderen im RAM-Cache 95 befindlichen Datenobjekte. Nachfolgend wird der Inhalt des RAM-Caches 95 durch einen einzigen zusammenhängenden Kopiervorgang 70 in die Speicherbank 80 umkopiert. Übersteigt der Speicherumfang der lebenden Datenobjekte die Größe des RAM-Caches 95, erfolgt das Umkopieren in Abhängigkeit von dessen Größe in mehreren Teilschritten. Nach jedem Teilschritt werden dann in der zuvor benutzten Speicherbank die noch vorhandenen Referenzen auf kopierte Datenobjekte durch Hinweisreferenzen auf die Adresse in der neuen Speicherbank ersetzt.

Eine weitere Ausführungsform zur Realisierung einer effektiven Speicherverwaltung ist in Fig. 4 veranschaulicht. Die in Fig. 4A gezeigte Struktur der Ausführungsform umfaßt die Register eines realen oder virtuellen Prozessor 100 sowie einen vorzugsweise im RAM eingerichteten Stapelspeicher, welcher in einen Operandenstapelspeicher 200 sowie einen Verwaltungsstapelspeicher 300 aufgeteilt ist.

Der reale/virtuelle Prozessor 100 verfügt über eine Vielzahl von Registern. Ein erstes Register 105 enthält dabei eine, im folgenden Instruktionsspointer (IP) genannte, Zeigervariable 105, welche auf einen von dem Chipkartencontroller aktuell abzuarbeitenden Programmbefehl zeigt. Ein weiteres Register 110 enthält eine, im folgenden Stackpointer (SP) genannte, zweite Zeigervariable, welche auf den Operandenstapelspeicher 200 gerichtet ist und darin auf den Operativdatenbereich derjenigen von dem Chipkartencontroller ausführbaren Methode anzeigt, die durch den aktuell abzuarbeitenden Programmbefehl aufgerufen ist. Unter Methode wird dabei ein Datencode verstanden, durch dessen Abarbeitung eine Funktion, z. B. eine Grundrechenart oder ein definierter Prozeß ausgeführt wird. Instruktionsspointer

105 und Stackpointer 110 zusammen bilden den Minimalinhalt eines Registersatzes 120 zur Beschreibung des Zustandes einer Methode. Der Block 120 kann, wie in Fig. 4A angedeutet, weitere Angaben umfassen, die dann entsprechend in weiteren Registern abgelegt sind. Ein weiteres Register 115 im Prozessor 100 enthält eine, im folgenden Returnpointer genannte, dritte Zeigervariable, welche auf den Verwaltungsstapelspeicher 300 gerichtet ist und dort den Beginn des Datenblocks anzeigt, der nach Abarbeitung des aktuell bearbeiteten Befehles, d. h. nach Ausführung der jeweils aktuell abzuarbeitenden Methode in das Prozessorregister 100 zu übernehmen ist.

Der Operandenstapelspeicher 200 enthält die zur Ausführung einer Methode erforderlichen Informationen. In einem ersten Speicherabschnitt 205 befinden sich dabei die einer Methode zugeordneten Parameter, in einem zweiten Abschnitt 210 die von einer Methode benutzten und erzeugten lokalen Variablen, in einem weiteren Abschnitt, dem Operativdatenbereich 215, die die Methode realisierenden Daten einschließlich der bei der Ausführung der Methode anfallenden Zwischenergebnisse. Der Operativdatenbereich 215 kann einen gesonderten Teilbereich 220 aufweisen, in dem, zunächst als Zwischenergebnis der ausgeführten Methode, Daten angelegt werden, welche zur Übernahme durch eine aufrufende weitere Methode bestimmt sind. Bei Aufruf der weiteren Methode bilden sie deren Parameter, der Teilbereich 220 wird mithin zum Parameterabschnitt der aufgerufenen Methode. In Wiederholung des vorbeschriebenen Anordnungsprinzips können sich im Operandenstapelspeicher 200 die Informationen zu weiteren Methoden befinden.

Der Verwaltungsstapelspeicher 300 dient zur vorübergehenden Aufnahme von Abbildern jeweils bestimmter Register des Prozessors 100, speziell eines den Zustand einer Methode beschreibenden Registersatzes 120 mit den Registern 105 und 110. Zu Beginn der Abarbeitung eines Programms durch den zugeordneten Chipkartencontroller ist der Verwaltungsstapelspeicher 300 in der Regel, wie in Fig. 4A angedeutet, nicht belegt.

Fig. 4B veranschaulicht die Funktion des Verwaltungsstapelspeichers 300. Er kommt zum Einsatz, wenn der Instruktionspointer 105 im Zuge einer Programmabarbeitung auf einen Befehl zeigt, welcher aus einer aktuell abgearbeiteten ersten Methode heraus eine weitere, zweite Methode aufruft. Vor Beginn der Bearbeitung der neu aufgerufenen, zweiten Methode werden die für die Weiterführung der Programmabarbeitung nach Ausführung der aufgerufenen Methode benötigten Register des Prozessors 100 gesichert. Dazu wird, wie durch den Pfeil 340 angedeutet, der den Zustand der aktuellen, aufrufenden Methode beschreibende Registersatz 120 aus dem Prozessor 100 in den Stapelspeicher 300 kopiert und dort als Registersatz 320 abgelegt. Mit dem Kopiervorgang wird zugleich eine Anpassung des Instruktionspointers 105 sowie des Stackpointers 110 vorgenommen, welche den Programmfortschritt sowie die Belegung des Stapelspeichers 200 nach Abarbeitung der Methode berücksichtigt. In seinem Abbild 305 wird der Instruktionspointer dabei durch Korrektur um einen Wert k (Meth. 1) so eingestellt, daß er auf den Programmteil zeigt, welcher dem Aufruf der zweiten Methode folgt. Das Abbild 310 des Stackpointers wird so eingerichtet, daß es entweder auf die Position 225 eines Parameters im Operandenstapelspeicher zeigt, an der sich ein von der aufgerufenen Methode gebildeter Rückgabewert R befindet, oder auf das jüngste Element des Operativdatenbereiches 215 der aufrufenden ersten Methode.

Ist der Zustand der aufrufenden ersten Methode im Verwaltungsstapelspeicher 300 gesichert, erfolgt, wie in Fig. 4C veranschaulicht, das Setzen der Register des Prozessors

100 zur Ausführung der aufgerufenen zweiten Methode. Der Stackpointer 110 wird dazu auf den Beginn des Operativdatenbereiches 245 der aufgerufenen zweiten Methode gestellt. Das Abbild 310 des Stackpointers im Verwaltungsstapelspeicher 300 zeigt gemäß der beim Kopieren vorgenommenen Voreinstellung auf einen Parameter 225 der zweiten Methode. Die Parameter 225, 230, 235 der aufgerufenen, zweiten Methode werden von der aufrufenden, ersten Methode angelegt und sind zunächst Bestandteil des Operativdatenbereiches 215 der aufrufenden, ersten Methode. Nach Beginn der Abarbeitung der aufgerufenen, zweiten Methode werden die Parameter 225, 230, 235 jedoch zu lokalen Variablen der zweiten Methode. Der Parameterabschnitt bildet auf diese Weise eine Überlappungsbereich zwischen aufrufender, erster und aufrufener, zweiter Methode.

Im Zuge der Abarbeitung der aufgerufenen, zweiten Methode fallen wiederum Zwischenergebnisse an, welche sich zunächst grundsätzlich im Operativdatenbereich 245 der aufgerufenen, zweiten Methode befinden. Liefert die zweite Methode einen Rückgabewert R, befindet sich dieser ebenfalls zunächst im Operativdatenbereich 245 der zweiten Methode. Spätestens bei Abschluß der Abarbeitung der zweiten Methode wird der Rückgabewert R dann an die oberste Position 225 in dem Parameterabschnitt 220 zwischen aufrufender erster und aufrufener, zweiter Methode übertragen. Auf die Top-Position 225 des Parameterabschnittes 220 zeigt auch das Abbild 310 des Stackpointers im Verwaltungsstapelspeicher 300.

Ist die Abarbeitung einer aufgerufenen Methode abgeschlossen, wird der im Verwaltungsstapelspeicher 300 angelegte, in angepaßter Form den Zustand der aufrufenden Methode angegebende Registersatz 320, wie durch den Pfeil 350 angedeutet, aus dem Verwaltungsstapelspeicher 300 in den Prozessor 100 zurückgeladen. Entsprechend der danach gegebenen Stellung des Stackpointers 105 wird nachfolgend die Programmabarbeitung entweder mit dem von der aufgerufenen Methode übergebenen Rückgabewert R oder mit dem jüngsten Element des Operativdatenbereiches der aufrufenden Methode fortgesetzt.

Es ist möglich, daß eine aufgerufene, zweite Methode ihrerseits wiederum eine weitere, dritte Methode aufruft, diese wiederum eine vierte u. s. w. Der Chipkartencontroller führt in diesem Fall jeweils dieselben Schritte aus wie bei erstmaligem Aufruf einer zweiten Methode aus einer bearbeiteten ersten Methode heraus. Er kopiert zunächst den den aktuellen Zustand der aufrufenden Methode wiedergebenden Registersatz aus dem Prozessor 100 unter Vornahme einer Anpassung des Instruktionspointers und des Stackpointers in den Verwaltungsstapelspeicher 300 und generiert dort einen Abbildblock 330, welcher unter dem Abbildblock 320 der zuletzt aufgerufenen Methode angeordnet wird. Sodann wird der Stackpointer im Prozessor 100 auf den Operativdatenbereich der neu aufgerufenen Methode gesetzt und die Abarbeitung der Methode eingeleitet. Die Abarbeitung wird abgeschlossen durch eine ggf. erfolgende Rückübertragung eines Rückgabewertes R an eine vorbestimmte Position im Parameterabschnitt der aufrufenden Methode und durch Rückladen des den Zustand der aufrufenden Methode angegebenden, angepaßten Registersatzes 330 aus dem Verwaltungsstapelspeicher in den Prozessor 100.

Das anhand der Fig. 4 beschriebene Prinzip läßt sich analog für weitere Methodenaufrufe aus aktuell bearbeiteten Methoden heraus fortsetzen.

Das anhand von Fig. 4 beschriebene Speicherverwaltungskonzept hat den Vorteil, daß der für die Ausführung eines Programmes benötigte Arbeitsspeicherraum auf physikalisch getrennte Speicher 200, 200, 300 verteilbar ist. Die technische Gestaltung der Speicher 100, 200, 300 läßt sich

dabei an die tatsächlichen Anforderungen anpassen. In Anwendungen mit physikalisch stark beschränktem Speicher-  
raum, d. h. besonders bei tragbaren Datenträgern in Form  
von Chipkarten, läßt sich dadurch die Effizienz der Pro-  
grammabarbeitung verbessern, indem der stark beschränkte  
schnelle Arbeitsspeicher nur mit tatsächlich einen schnellen  
Speicher erfordernden Operationen belastet wird.

#### Patentansprüche

1. Speicheranordnung für einen ein- und abschaltbaren  
tragbaren Datenträger, welcher zur Ausführung von  
transiente Datenobjekte erzeugenden Datenträgerpro-  
grammen ausgebildet ist und das Nachladen solcher  
Datenträgerprogramme gestattet, mit
  - einer ersten Speichereinrichtung (30) zur flüch-  
tigen Datenspeicherung,
  - einer zweiten Speichereinrichtung (15) zur  
nichtflüchtigen Datenspeicherung,
  - wobei die erste Speichereinrichtung (30) min-  
destens einen Speicherbereich (10) aufweist, in  
dem sich Informationen (20a, 20b, 20c, 20d) be-  
finden, welche Lage und Größe von in der zweiten  
Speichereinrichtung (15) für die Ausführung we-  
nigstens eines Kartenprogrammes eingerichteten  
Speicherbereichen (25a, 25b, 25c, 25d) angeben,  
und
  - wobei eine der Speichereinrichtung (15) zuge-  
ordnete Programmroutine die in dem Speicherbe-  
reich (10) befindlichen Informationen (20a, 20b,  
20c, 20d) im Anschluß an einem Abschaltvor-  
gang, welcher eine Löschung der Dateninhalte der  
ersten Speichereinrichtung (30) zur Folge hat, so  
neu festlegt, daß die von ihnen bezeichneten Spei-  
cherbereiche (25a, 25b, 25c, 25d) in der ersten  
Speichereinrichtung (30) einen zusammenhän-  
genden Speicherbereich belegen.
2. Speicheranordnung nach Anspruch 1, dadurch ge-  
kennzeichnet, daß die Neufestlegung der Informa-  
tionen (20a, 20b, 20c, 20d) jeweils unmittelbar nach dem  
Wiedereinschalten eines abgeschalteten Datenträgers  
erfolgt.
3. Speicheranordnung nach Anspruch 1, dadurch ge-  
kennzeichnet, daß die die Neufestlegung der Informa-  
tionen (20a, 20b, 20c, 20d) durchführende Program-  
mroutine Teil des Betriebssystems des Datenträgers ist.
4. Tragbarer, ein- und abschaltbarer Datenträger, ge-  
kennzeichnet durch eine Speicheranordnung nach ei-  
nem der Ansprüche 1 bis 3.
5. Verfahren zur Verwaltung der Speicheranordnung  
eines ein- und abschaltbaren, tragbaren Datenträgers,  
der zur Ausführung von Datenträgerprogrammen aus-  
gebildet ist, welche in einem vorzugsweise flüchtigen  
Speicher transiente Datenobjekte anlegen, deren Lage  
und Größe durch in einem nichtflüchtigen Speicher ab-  
gelegte Informationen festgehalten wird, darin beste-  
hend, daß
  - unmittelbar nach den Einschalten und/oder un-  
mittelbar vor dem Ausschalten des Datenträgers  
die in der zweiten Speichereinrichtung festgehal-  
tenen Informationen (20a, 20b, 20c, 20d) neu be-  
stimmt werden, wobei die Neubestimmung so er-  
folgt, daß die durch die Informationen (20a, 20b,  
20c, 20d) bezeichneten Speicherbereiche (25a,  
25b, 25c, 25d) der ersten Speichereinrichtung (30)  
einen zusammenhängenden Speicherbereich bele-  
gen.
6. Verfahren nach Anspruch 5, dadurch gekennzeich-

net, daß der Einschaltsschritt durch Verbinden mit einer  
Betriebsstromquelle bewirkt wird.

7. Speicheranordnung für einen ein- und abschaltbaren  
tragbaren Datenträger, welcher zur Ausführung von  
transiente Datenobjekte anlegenden Datenträgerpro-  
grammen ausgebildet ist und das Nachladen solcher  
Datenträgerprogramme gestattet, mit:

- einer ersten Speichereinrichtung mit einem er-  
sten Speicherbereich (80), sowie mit einer zwei-  
ten Speichereinrichtung mit einem zweiten Spei-  
cherbereich (90), welcher jeweils zur Aufnahme  
von mindestens einem, in Abhängigkeit vom Ab-  
lauf eines Datenträgerprogrammes einen aktiven  
oder einen passiven Zustand annehmenden Daten-  
objektes (45a, 45b, 45c, 45d) vorgesehen ist, wo-  
bei die Ausführung eines Datenträgerprogrammes  
zunächst jeweils unter Nutzung einer der Spei-  
chereinrichtungen (80, 90) erfolgt,
- Mitteln zum Wechseln der für die Ausführung  
eines Datenträgerprogrammes genutzten Spei-  
chereinrichtung auf die jeweils andere Speicher-  
einrichtung auf ein vordefiniertes Ereignis hin,
- wobei der Wechsel der Speichereinrichtung  
eine der Speicheranordnung zugeordnete Kom-  
paktierungsroutine in Gang setzt, welche gemäß  
einem üblichen Garbage-Collection-Verfahren  
aus dem ersten Speicherbereich (80) selektiv alle  
aktiven Datenobjekte (45b, 45c) ermittelt und  
diese zusammenhängend in den Speicherbereich  
(90) der zweiten Speichereinrichtung umkopiert.

8. Speicheranordnung nach Anspruch 7, dadurch ge-  
kennzeichnet, daß die erste (80) und zweite Spei-  
chereinrichtung (90) als erste und zweite Speicherbank in  
einer gemeinsamen Speichervorrichtung ausgebildet  
sind.

9. Speicheranordnung nach Anspruch 7, dadurch ge-  
kennzeichnet, daß sie eine Zwischenspeichereinrich-  
tung (95) aufweist, in welche die Kompaktierungsrou-  
tine die aktiven Datenobjekte (45b, 45c) ablegt.

10. Verfahren zur Verwaltung der Speicheranordnung  
eines tragbaren Datenträgers, die wenigstens einen er-  
sten (80) sowie einen zweiten Speicherbereich (90)  
aufweist, welche jeweils zur Aufnahme von minde-  
stens einem in Abhängigkeit vom Ablauf eines Daten-  
trägerprogrammes einen aktiven oder einen passiven  
Zustand annehmenden Datenobjektes (45a, 45b, 45c,  
45d) vorgesehen sind,

mit folgenden Schritten:

- Ausführen eines Datenträgerprogrammes unter  
Nutzung eines der Speicherbereiche (80, 90),
- auf ein definiertes Ereignis hin Wechseln des  
benutzten Speicherbereiches mit nachfolgender  
Ausführung eines Datenträgerprogrammes unter  
Verwendung des zuvor nicht benutzten Speicher-  
bereiches,
- wobei der Speicherbereichswechsel erfolgt, in-  
dem alle aktiven Datenobjekte (45b, 45d) gemäß  
einem üblichen Garbage-Collection-Verfahren  
aus dem ersten (80) in den zweiten Speicherbe-  
reich (90) umkopiert,
- alle Datenobjekte (45a, 45b, 45c, 45d) des er-  
sten Speicherbereiches (80) inaktiviert und
- alle Datenobjekte des zweiten Speicherberei-  
ches (90) aktiviert werden.

11. Verfahren nach Anspruch 10, dadurch gekenn-  
zeichnet, daß die kopierten Datenobjekte (45b, 45d) im  
zweiten Speicherbereich (90) in einem zusammenhän-  
genden Teilbereich (60a) angeordnet werden.



12. Verfahren nach Anspruch 10, dadurch gekennzeichnet, daß die Datenobjekte (45b, 45d) zunächst in einen hinsichtlich der Zugriffszeiten schnellen Zwischenspeicher (95) kopiert werden und dort die Neuordnung gemäß dem eingesetzten Garbage-Collection-Verfahren erfolgt. 5
13. Speicheranordnung für einen Datenträger mit
- einem Prozessor (100) zugehörigen Registern, die zumindest eine Zeigervariable (105) zur Bezeichnung einer Programminstruktion eines abzuarbeitenden Programmes sowie eine Zeigervariable (110) zur Bezeichnung des Operativdatenbereiches (215) einer im Rahmen des Programmes aufrufbaren Methode beinhalten, 10
  - einen ersten Stapelspeicher (200) zur Speicherung der Methodendaten wenigstens einer im Zuge der Abarbeitung eines Programmes aufrufbaren Methode, wobei die Methodendaten einen Operativdatenbereich (215) mit der Ausführbarkeit der Methode bewirkenden Daten und einen Parameterabschnitt (205) zur Aufnahme von bei der Ausführung der Methode benötigten Parametern aufweisen; sowie 15
  - einen zweiten Stapelspeicher (300) zur vorübergehenden Aufnahme von den Zustand der aufrufenden Methode beschreibenden Registern aus dem Prozessor (100), wenn von einer im Rahmen der Programmabarbeitung aktuell abgearbeiteten Methode eine weitere Methode aufgerufen wurde. 20
14. Speicheranordnung nach Anspruch 13, dadurch gekennzeichnet, daß die Methodendaten weiterhin einen Abschnitt (205, 220) aufweisen, in dem sich von der Methode benutzte lokale Variablen befinden. 25
15. Speicheranordnung nach Anspruch 13, dadurch gekennzeichnet, daß die den Zustand einer aufrufenden Methode beschreibenden Daten zumindest eine Zeigervariable (305) zur Bezeichnung einer Programminstruktion des abzuarbeitenden Programmes sowie eine Zeigervariable (310) zur Bezeichnung einer Position im Operativdatenbereich (215) der aufrufenden Methode umfassen. 30
16. Speicheranordnung nach Anspruch 13, dadurch gekennzeichnet, daß der erste und der zweite Stapelspeicherbereich (200, 300) in separaten Speicherstrukturen ausgebildet sind. 35
17. Tragbarer Datenträger, gekennzeichnet durch eine Speicheranordnung gemäß einem der Ansprüche 13 bis 16.
18. Verfahren zum Aufrufen einer Methode aus einer im Zuge der Abarbeitung eines Programmes bearbeiteten anderen Methode heraus in einer prozessorgesteuerten Speicheranordnung mit einem Prozessor zugeordneten Registern, die zumindest eine Zeigervariable zur Bezeichnung einer Programminstruktion sowie eine Zeigervariable zur Bezeichnung des Operativdatenbereiches einer Methode beinhalten, sowie einem Stapelspeicher zur Speicherung von Methodendaten, mit folgenden Schritten: 40
- Anordnen der Methodendaten der aufrufenden und der aufgerufenen Methode in einem ersten Teil (200) des Stapelspeichers, wobei die Methodendaten jeweils in einen Operativdatenbereich (215) mit die Ausführbarkeit der Methode bewirkenden Daten sowie einen Parameterabschnitt (205) zur Aufnahme von bei der Ausführung benötigten Parametern gegliedert werden; 45
  - Einrichten eines zweiten Teiles (300) des Sta-

- pelspeichers zur vorübergehenden Aufnahme von den Zustand einer aufrufenden Methode beschreibenden Daten aus dem Register im Prozessor (100),
- Übertragen der den Zustand einer aufrufenden Methode beschreibenden Register aus dem Prozessor (100) in den zweiten Teil des Stapelspeichers (300), wenn eine zweite Methode aus einer ablaufenden ersten Methode heraus aufgerufen wurde,
  - Ausführen der aufgerufenen Methode,
  - Rückübertragen der in den zweiten Teil des Stapelspeichers (300) übertragenen Register in den Prozessor (100) nach Abschluß der Ausführung der aufgerufenen Methode.
29. Verfahren nach Anspruch 18, dadurch gekennzeichnet, daß die den Zustand der aufrufenden Methode beschreibenden Daten beim Übertragen in den zweiten Teil des Stapelspeichers (300) an die Abarbeitungssituation nach Ausführung der aufgerufenen Methode angepaßt werden.
20. Verfahren nach Anspruch 18, dadurch gekennzeichnet, daß die im Parameterabschnitt (220) befindlichen Parameter von der aufrufenden Methode angelegt werden.
21. Verfahren nach Anspruch 19, dadurch gekennzeichnet, daß, falls die aufgerufene Methode einen Rückgabewert (R) liefert, dieser auf einer vorbestimmten Position (225) im Parameterabschnitt (205) der aufrufenen Methode abgelegt wird.
22. Verfahren nach Anspruch 21, dadurch gekennzeichnet, daß die den Zustand der aufrufenden Methode beschreibenden Daten zumindest eine Zeigervariable (310) beinhalten, welche auf die vorbestimmte, den Rückgabewert enthaltende Position im Parameterabschnitt (205) zeigt.

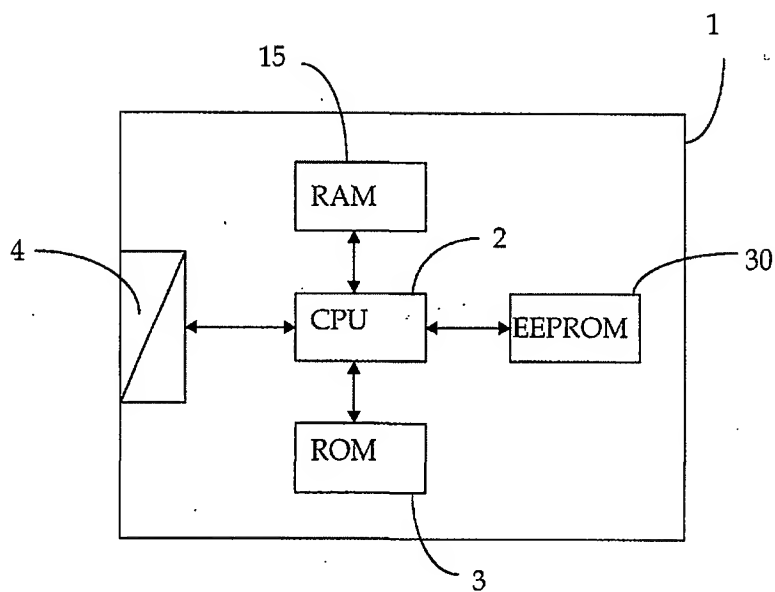
---

Hierzu 6 Seite(n) Zeichnungen

---



- Leerseite -



**Fig.1**

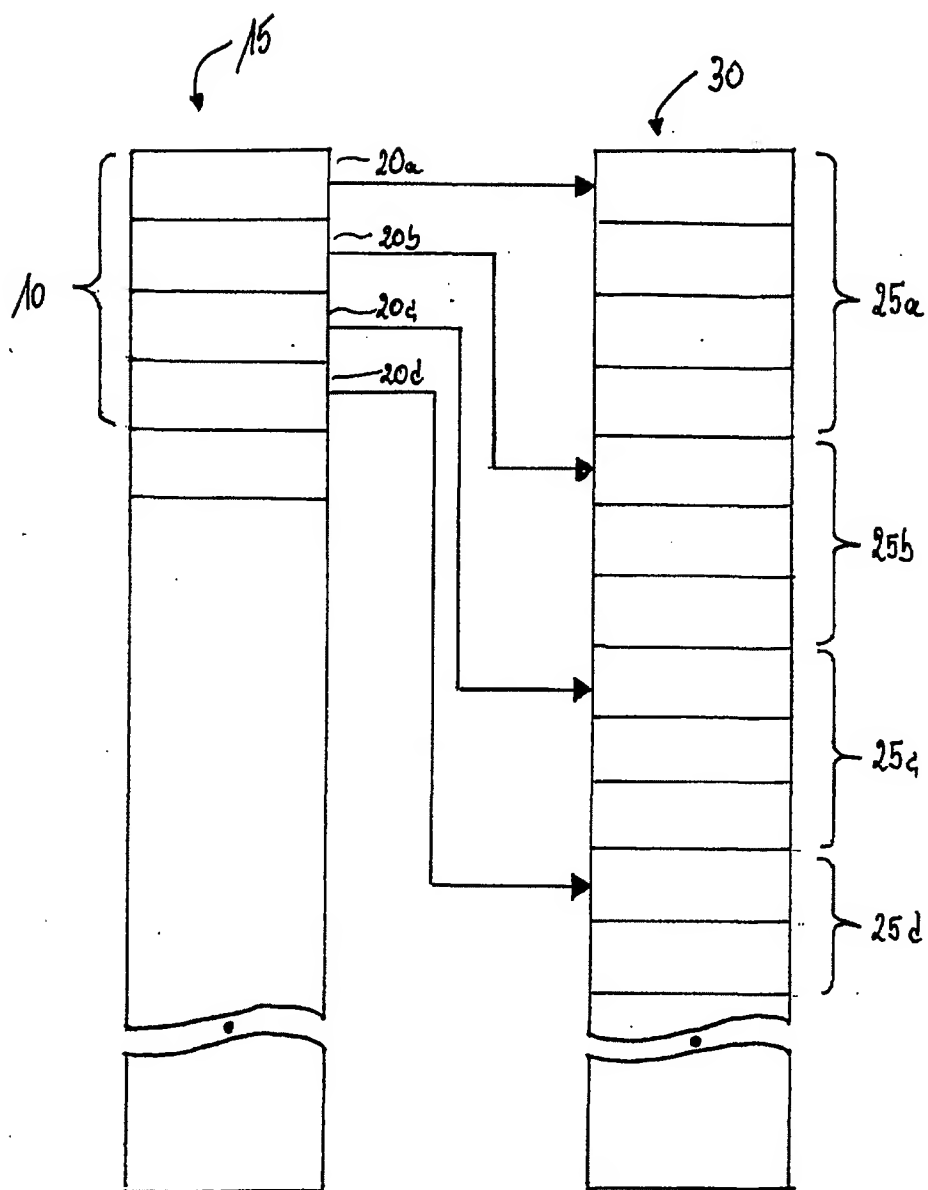


Fig. 2A

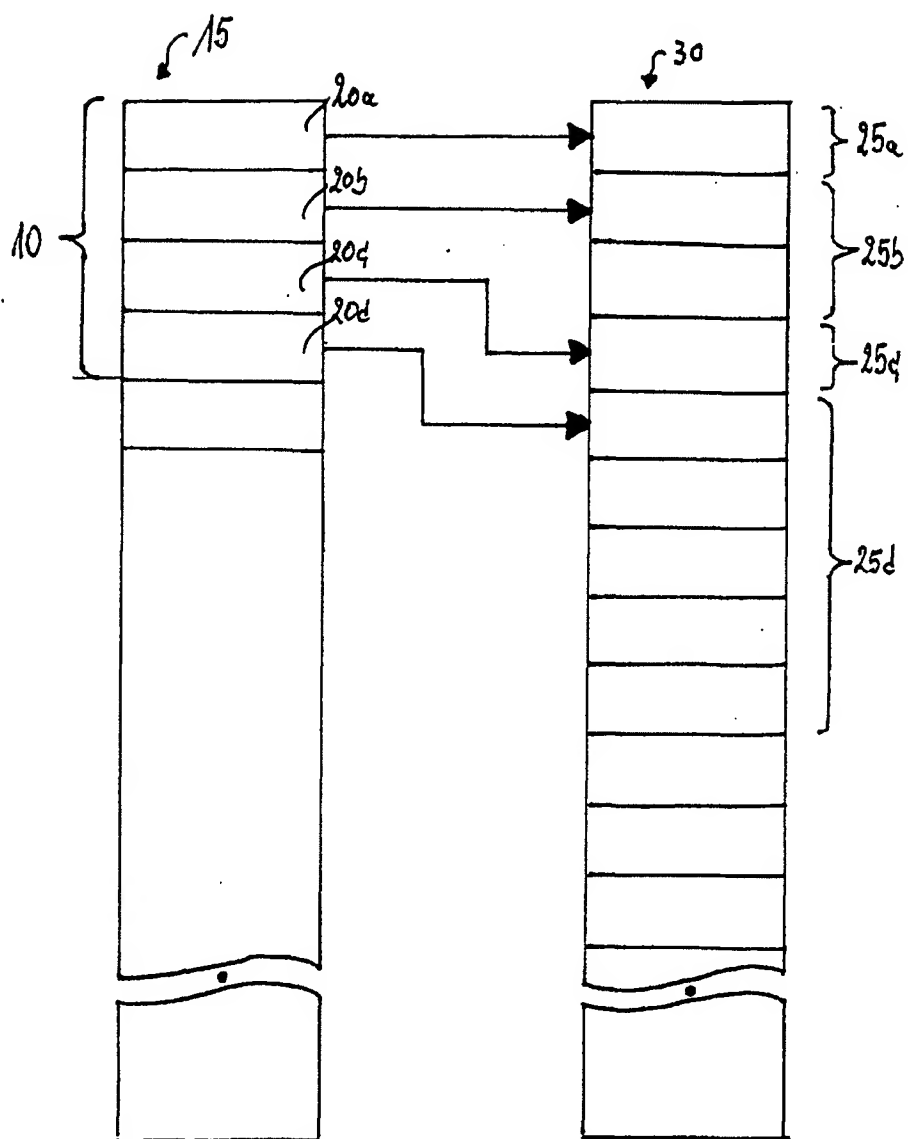


Fig. 2B

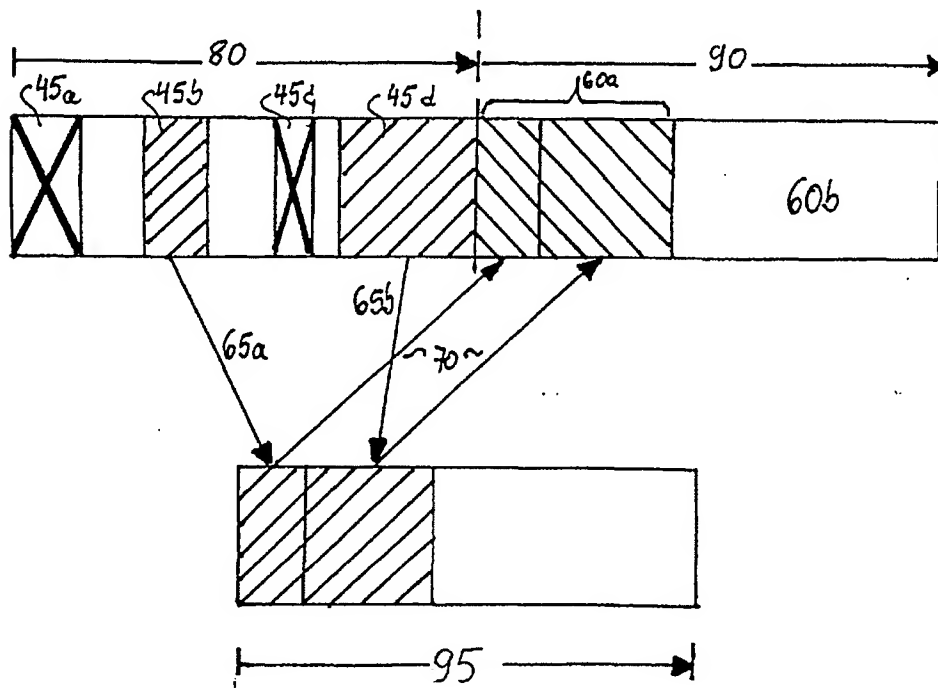
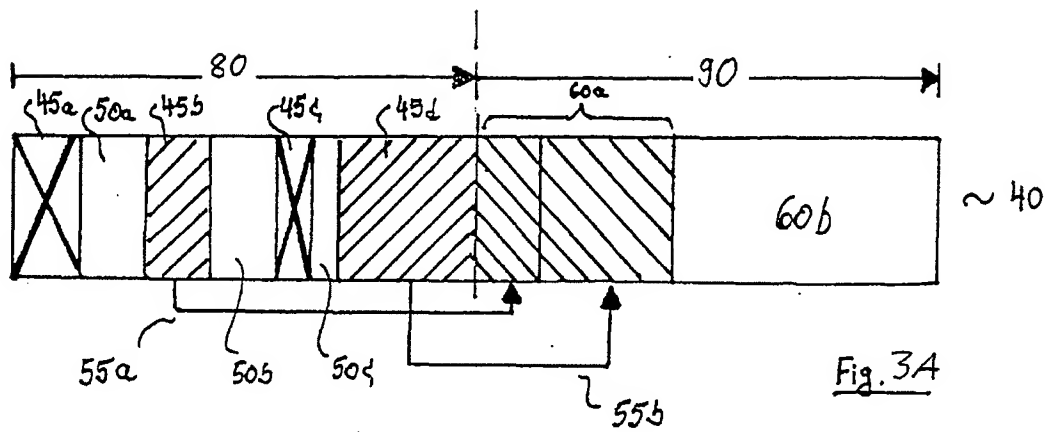


Fig. 3B

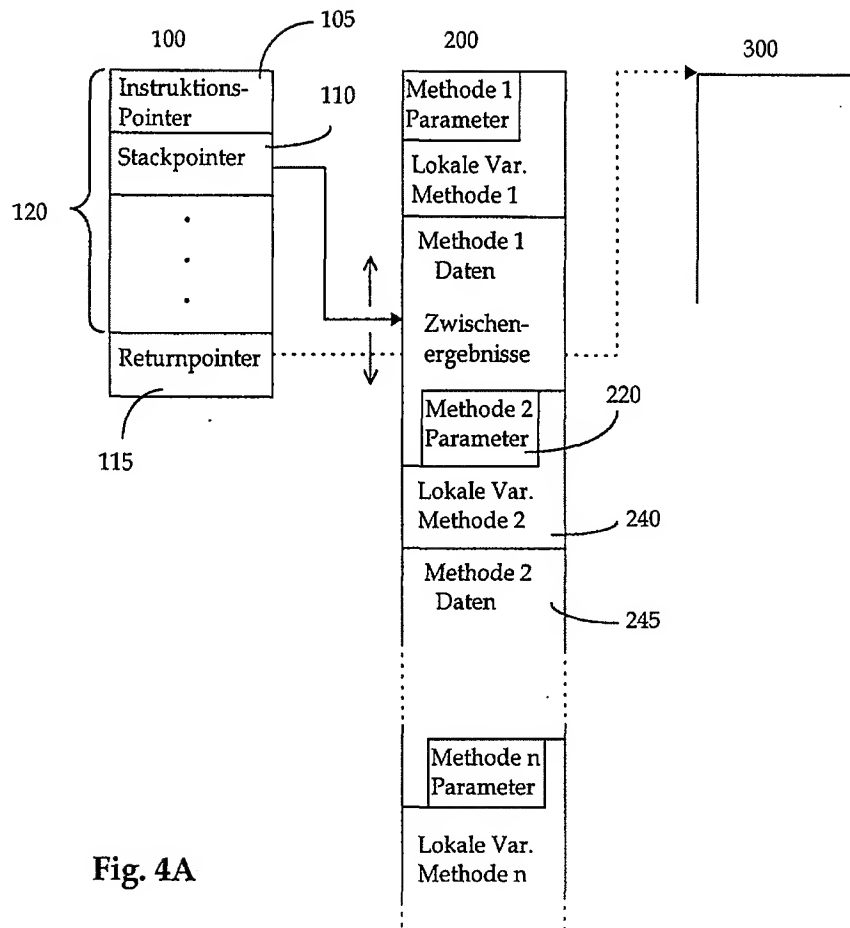


Fig. 4A

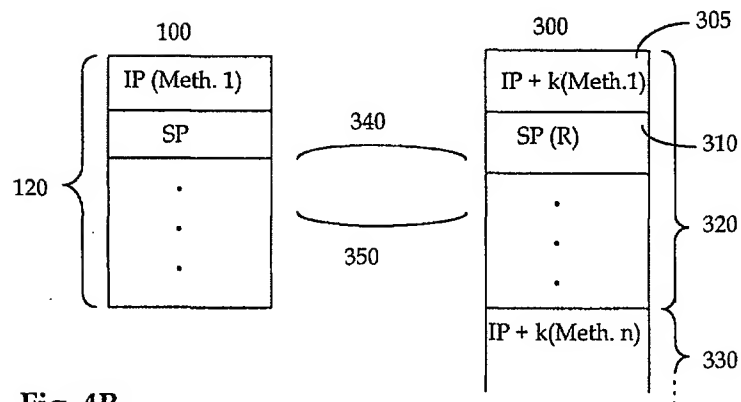


Fig. 4B

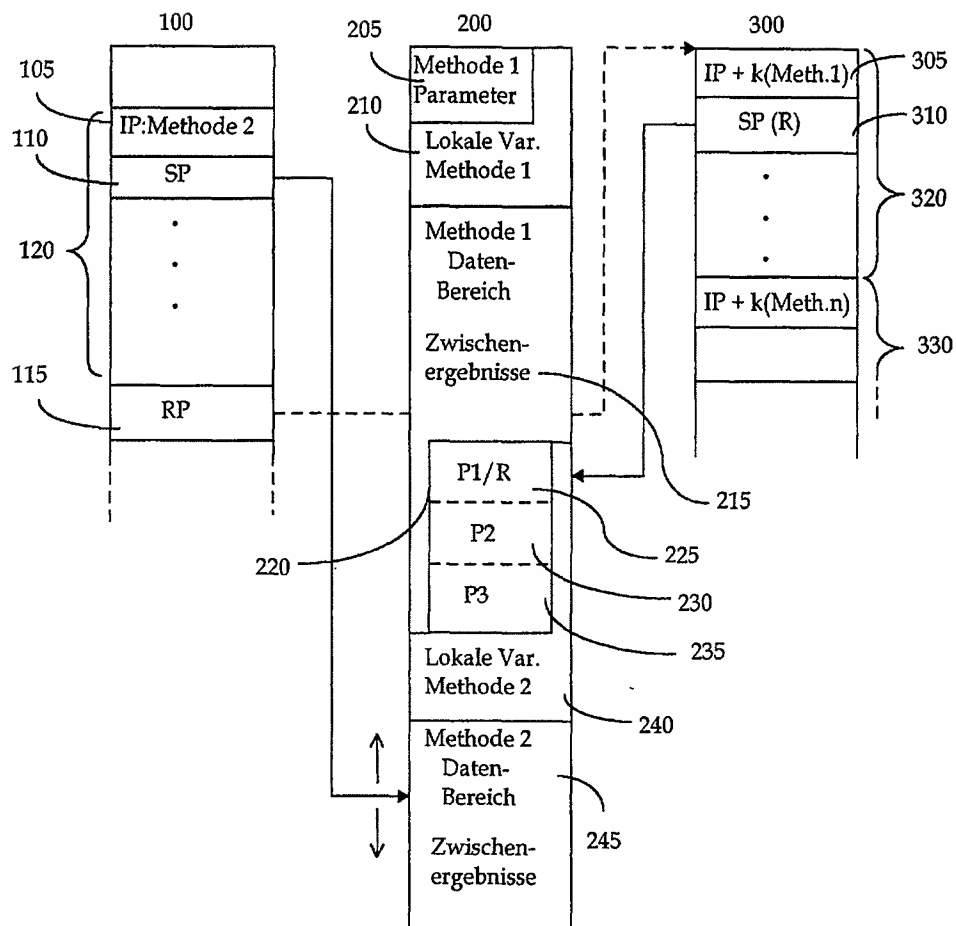


Fig. 4C